Computer Architecture - tutorial 5 [SOLUTIONS]

Context, Objectives and Organization

The goal of the quantitative exercise in this tutorial is to explore qualitatively and quantitatively some hardware and software optimizations to improve cache performance.

E1: CAR September 2003 exam P2 - groups of 2 - 35 min

Problem

Consider a computer system with a first-level data cache with the following characteristics: size: 16KBytes; associativity: direct-mapped; line size: 64Bytes; addressing: physical.

The system has a separate instruction cache and you can ignore instruction misses in this problem. This system is used to run the following code:

```
for (i=0; i<4096; i++)
  X[i] = X[i] * Y[i] + C
```

Assume that both X and Y have 4096 elements, each consisting of 4 bytes (single precision floating point). These arrays are allocated consecutively in physical memory. The assembly code generated by a naive compiler is the following:

```
loop: lw f2, 0(r1)       # load X[i]
      lw f4, 0(r2)       # load Y[i]
      multd f2, f2, f4   # perform the multiplication
      addd f2, f2, f0    # add C (in f0)
      sw 0(r1), f2       # store the new value of X[i]
      addi r1, r1, 4     # update address of X
      addi r2, r2, 4     # update address of Y
      addi r3, r3, 1     # increment loop counter
      bne r3, 4096, loop # branch back if not done
```

a. How many data cache misses will this code generate? Breakdown your answer into the three types of misses. What is the data cache miss rate?

b. Provide a software solution that significantly reduces the number of data cache misses. How many data cache misses will your code generate? Breakdown the cache misses into the three types of misses. What is the data cache miss rate?

c. Provide a hardware solution that significantly reduces the number of data cache misses. You are free to alter the cache organization and/or the processor. How many data cache misses will your code generate? Breakdown the cache misses into the three types of misses. What is the data cache miss rate?

Solution

a.

Misses:

$coldmisses : 2 * (4096/16) = 512$

as every 16th iteration will miss on the load to X and to Y, for the first time

$conflictmisses : 4096 + (15/16) * 4096 = 4096 + 3840 = 7936$

as every store to X will miss because it was displaced by the previous load to Y, and as every load to the remaining 15 elements of Y in a line will miss because they were displaced by the previous store to X

$total : 512 + 7936 = 8448$ misses

So:

$missrate : 8448/12288 = 0.6875$ (68.75%)

as there are $3 * 4096 = 12288$ memory references in the loop

b.

- alternative 1: merge arrays X and Y, interleaving their elements, now 8 elements of each array fit in the same line

  $coldmisses : 4096/8 = 512$

  as every 8th iteration will miss on the load to X

  $conflictmisses : 0$

  $total : 512$ misses

  So:

  $missrate : 512/12288 = 0.0417$ (4.17%)

- alternative 2: separate arrays X and Y in memory by a distance not multiple of 16K

  $coldmisses : 2 * (4096/16) = 512$

  as every 16th iteration will miss on the load to X and on the load to Y

  $conflictmisses : 0$

  $total : 512$ misses

  So:

  $missrate : 512/12288 = 0.0417$ (4.17%)

c.

- alternative 1: double the cache size to 32KB

  $coldmisses : 2 * (4096/16) = 512$

  as every 16th iteration will miss on the load to X and on the load to Y

  $conflictmisses : 0$

  $total$: 512 misses

  So:

  $missrate : 512/12288 = 0.0417$ (4.17%)

- alternative 2: make the cache set associative

  $coldmisses : 2 * (4096/16) = 512$

  as every 16th iteration will miss on the load to X and on the load to Y

  $conflictmisses : 0$

  $total$: 512 misses

  So:

  $missrate : 512/12288 = 0.0417$ (4.17%)

- alternative 3: increase the block size by z times

  $coldmisses : 4096/(16 * z)$

  as every 16*z'th iteration will miss on the load to X (in the limit we use 16KByte lines and the have only a single cold miss)

  $conflictmisses : 2 * 4096 = 8192$

  as every load to Y will cause a conflict with the just-loaded line of X and every store to X will cause a conflict with the just-loaded line of Y

- alternative 4: add a next-line prefetcher

  $coldmisses : 2 * (4096/32) = 256$

  in effect, the next-line prefetcher cuts the cold misses in 2 (each cold miss eliminates the very next cold miss).

  $conflictmisses : 4096 + (31/32) * 4096 = 8064$

  So:

  $missrate : (256 + 8064)/12288 = 0.667$ (66.7%)

  Note that a streaming prefetcher would eliminate all but the first cold miss on each of the arrays.

- alternative 5: add a victim cache

  $coldmisses$: $4096/16 = 256$

  as every 16th iteration will miss on the load to X (the other 15 iterations reuse the line left over from the store to X)

  $conflictmisses$: 4096

  as every load to Y will cause a conflict with the just loaded line of X (the stores to X now hit in the victim cache)

  $total$: $256 + 4096 = 4352$ misses

  So:

  $missrate$: $4352/12288 = 0.3542$ (35.42%)

Boris Grot 2018. Thanks to Vijay Nagarajan, Nigel Topham and Marcelo Cintra.